1 Introduction

CS 189 / 289A [Spring 2024]
Machine Learning
Jonathan Shewchuk

https://people.eecs.berkeley.edu/~jrs/189/

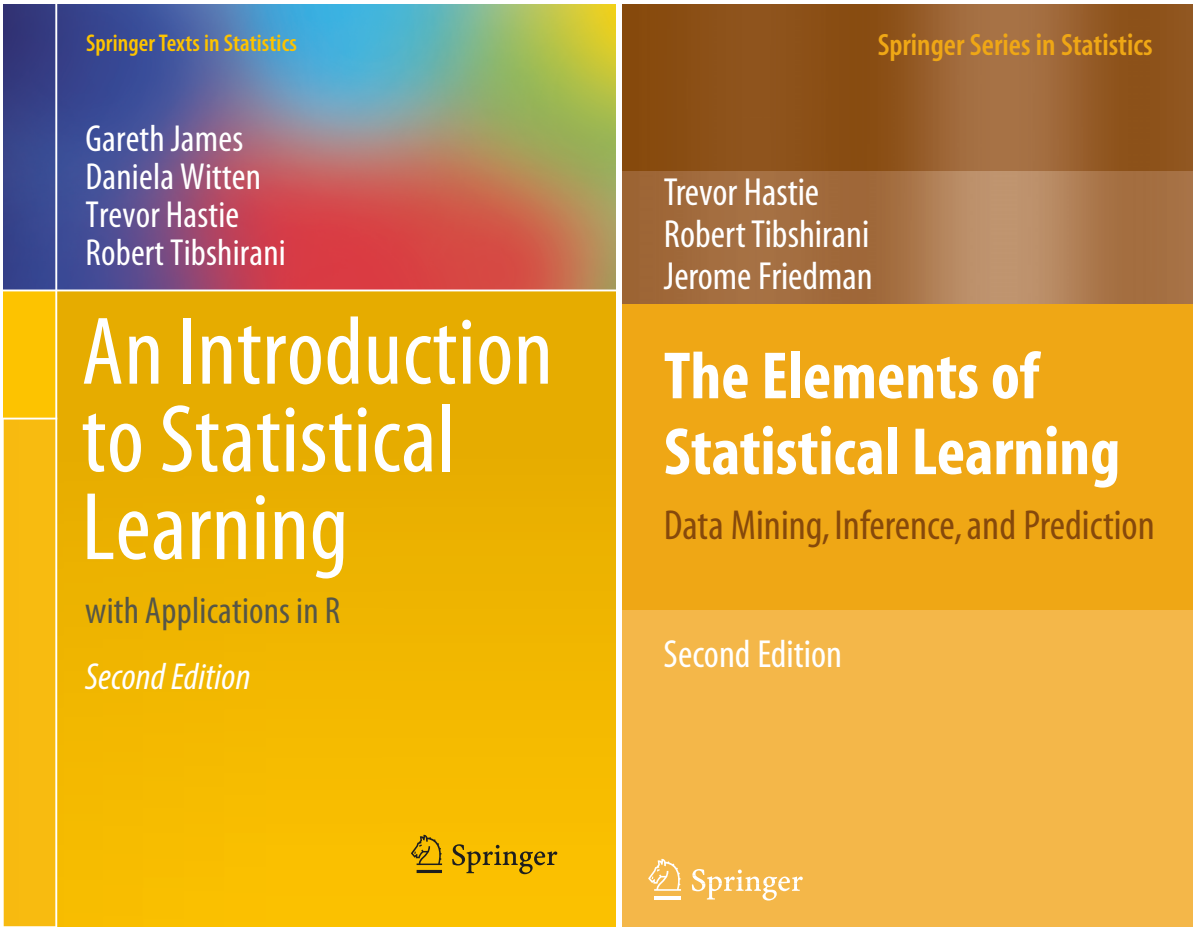
Homework 1 due next Wednesday.

Questions: Please use Ed Discussion, not email. [Ed Discussion has an option for private questions, but please use public for most questions so other people can benefit.]
For personal matters only, jrs@berkeley.edu

Discussion sections (Tue & Wed):
Attend any section. [We'll put up a list on Ed Discussion.]
[We might have a few advanced sections, including research discussion or exam problem preparation.]
Sections start Tuesday. [Next week.]

[Enrollment: 703 students max. 452 waitlisted. Expecting many drops. EECS grads have highest priority; CD/DS undergrads second; non-EECS grads third; a few concurrent enrollment students will be admitted.]

[Textbooks: Available free online. Linked from class web page.]



Prerequisites

Vector calculus: Math 53 [or another vector calculus course]
Linear algebra: Math 54, Math 110, or EE 16A+16B [or another linear algebra course]
Probability: CS 70, EECS 126, or Stat 134 [or another probability course]
Plentiful programming experience [TAs have no obligation to look at your code.]
NOT CS 188

Grading: 189

40% 7 Homeworks. Late policy: 5 slip days total
20% Midterm: TBA, 6:30–8:30 PM
40% Final Exam: Friday, May 10, 3–6 PM

Grading: 289A

40% HW
20% Midterm
20% Final
20% Project

Cheating

- Discussion of HW problems is encouraged. Showing other students *small* amounts of code is okay.
- All homeworks, including programming, must be written individually.
- We will actively check for plagiarism.
- Typical penalty is a large NEGATIVE score, but I reserve right to give an instant F for even one violation, and will always give an F for two.

[Last time I taught CS 61B, we had to punish roughly 100 people for cheating. It was very painful. Please don't put me through that again.]

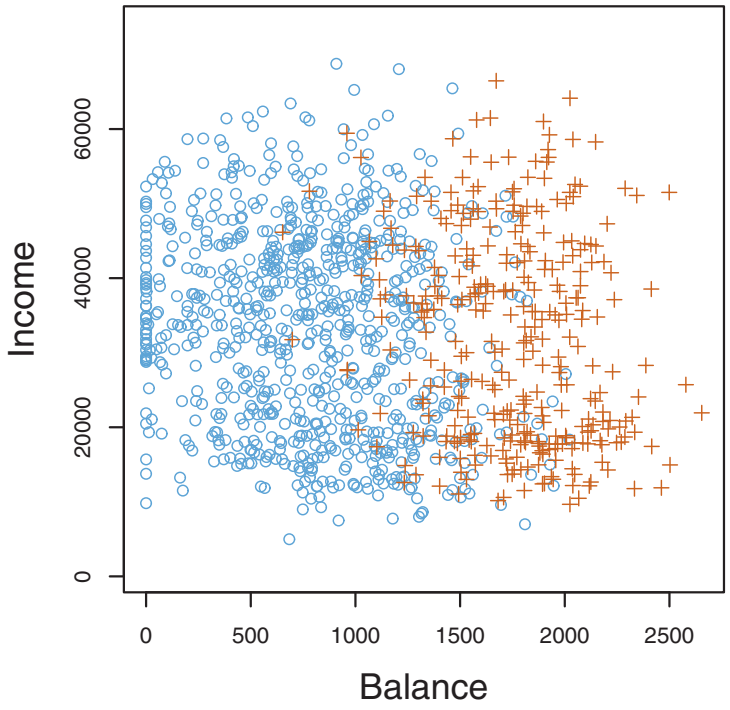
CORE MATERIAL

- Finding patterns in data; using them to make predictions.
- Models and statistics help us understand patterns.
- Optimization algorithms "learn" the patterns.

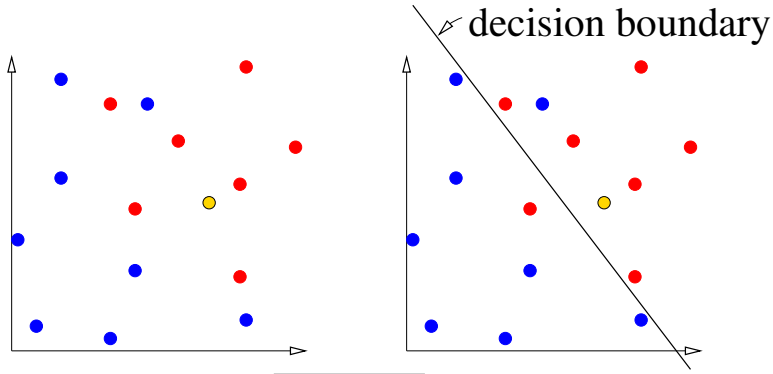
[The most important part of this is the data. Data drives everything else.
You cannot learn much if you don't have enough data.
You cannot learn much if your data has bad quality.
But it's amazing what you can do if you have lots of good data.
Machine learning has changed a lot in the last two decades because the internet has made truly vast quantities of data available. For instance, with a little patience you can download tens of millions of photographs. Then you can build a 3D model of Paris.
Some techniques that had fallen out of favor, like neural networks, have come back big in recent years because researchers found that they work so much better when you have vast quantities of data.]

CLASSIFICATION

- Collect training points with class labels: reliable debtors & defaulted debtors
- Evaluate new applicants—predict their class

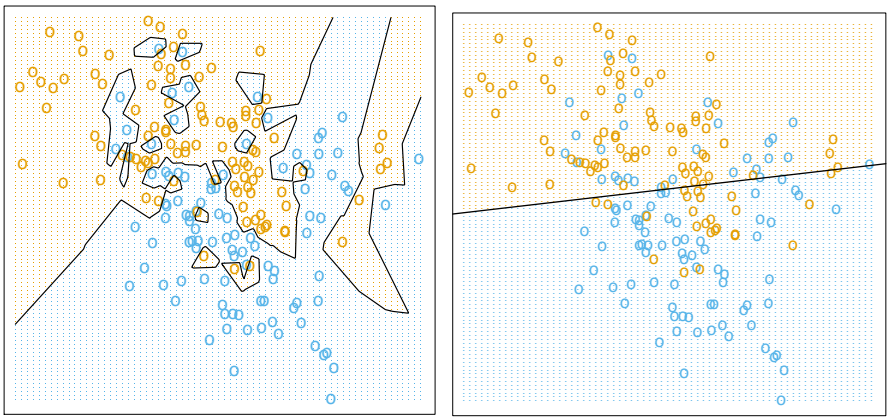


creditcardscrop.pdf (ISL, Figure 4.1) [The problem of classification. We are given data points, each belonging to one of two classes: orange crosses represent people who defaulted on their credit cards, and blue circles represent those who didn't. Then we are given additional points whose class is unknown, and we are asked to predict what class each new point is in. Given the credit card balance and annual income of new applicants, predict whether they will default on their debt.]



[Draw this figure by hand. classify.pdf]
[Draw 2 colors of dots, almost but not quite linearly separable.]
["How do we classify a new point?" Draw a point in a third color.]
[One possibility: look at its nearest neighbor.]
[Another possibility: draw a linear decision boundary; label it.]
[Those are two different *models* for the nature of this data.]

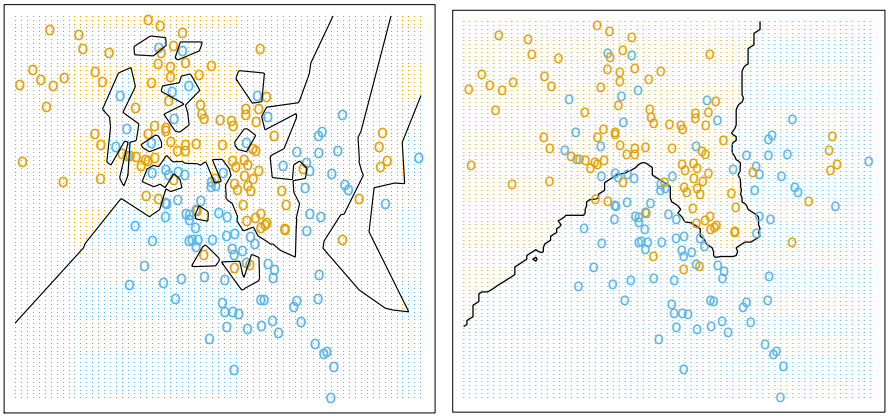
[We'll learn some ways to compute linear decision boundaries in the next several lectures. But for now, let's compare these two methods.]



classnear.pdf, classlinear.pdf (ESL, Figures 2.3 & 2.1) [Two examples of classifiers for the same data: a nearest neighbor classifier (left) and a linear classifier (right). The decision boundaries are in black.]

[At left we have a *nearest neighbor classifier*, which classifies a new point by finding the nearest point in the training data, and assigning it the same class. At right we have a *linear classifier*, which guesses that everything above the line is brown, and everything below the line is blue. At right, the linear decision boundary—the black line—is explicitly computed by the classifier. At left, the decision boundary is *not* computed; the classifier just takes a new point and computes the distances to all the training points.]

[The neighbor classifier at left has a big advantage: it classifies all the training data correctly, whereas the linear classifier does not. But the linear classifier has an advantage too. Somebody please tell me what.]

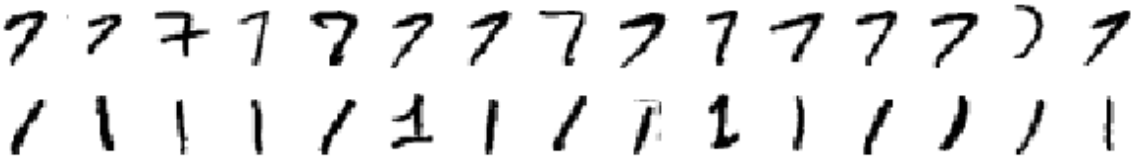


classnear.pdf, classnear15.pdf (ESL, Figures 2.3 & 2.2) [A 1-nearest neighbor classifier and a 15-nearest neighbor classifier.

[The *15-nearest neighbor classifier* classifies a new point by looking at its 15 nearest neighbors and letting them vote for the correct class.]

[The left figure is an example of what's called overfitting. In the left figure, observe how intricate the decision boundary is that separates the positive examples from the negative examples. It's a bit too intricate to reflect reality. In the right figure, the decision boundary is smoother. Intuitively, that smoothness is probably more likely to correspond to reality.]

Classifying Digits



sevensones.pdf [In the MNIST digit recognition problem, we are given handwritten digits, and we are asked to learn to distinguish them. See Homework 1.]

Express these images as vectors

$$\begin{bmatrix} 3 & 3 & 3 & 3 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 3 & 3 & 3 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \\ 0 \\ 0 \\ 2 \\ 3 \\ 0 \\ 0 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}$$

Images are points in 16-dimensional space. Linear decision boundary is a hyperplane.

TRAIN, VALIDATE, TEST

How we classify:

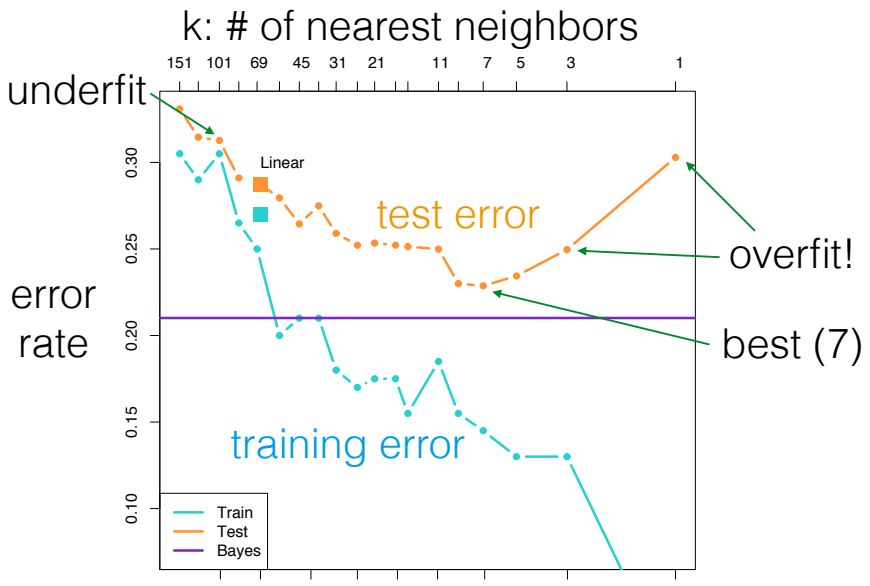
- We are given labeled data—sample points with class labels.
- Hold back a subset of the labeled points, called the validation set. Maybe 20%. The other 80% is the training set. [Warning: the term *training data* is not used consistently. Often "training data" refers to *all* the labeled data. You have to judge from context.]
- Train one or more classifiers: they learn to distinguish 7 from not 7. Use training set to learn model weights. Do NOT use validation set to train!!!
- Usually, train multiple learning algorithms, or one algorithm with multiple hyperparameter settings, or both [using the same training set for each].
- Validate the trained classifiers on the validation set. Choose classifier/hyperparameters with lowest validation error. Called validation. [When we do validation, we are not learning any more. We are checking what classes our trained classifiers assign to our validation set, and counting how often they're right. We use this to judge our models—not how well they remember the training set labels.]
- Optional: Test the best classifer on a test set of NEW data. Final evaluation. Typically you do NOT have the labels. [But somebody else might have them, and assign you a score!]

[When I underline a word or phrase, that usually means it's a definition. My advice to you is to memorize the definitions I cover in class.]

3 kinds of error:

- Training error: fraction of training set not classified correctly. [This is zero with the 1-nearest neighbor classifier, but nonzero with the 15-nearest neighbor and linear classifiers. But that doesn't mean the 1-nearest neighbor classifier is always better. Remember that you cannot include the validation data in this calculation, even if somebody calls it "training data."]
- Validation error: fraction of validation set misclassified. Use this to choose classifier/hyperparameters. [You didn't use the validation set to train, so even the 1-nearest neighbor classifier can classify these points wrong. Validation error is almost always higher than training error.]
- Test error: fraction of test set misclassified. Used to evaluate **you**.

Most ML algorithms have a few hyperparameters that control over/underfitting, e.g. k in k-nearest neighbors.



overfitlabeled.pdf (modified from ESL, Figure 2.4)

- overfitting: when the validation/test error deteriorates because the classifier becomes too sensitive to outliers or other spurious patterns.
- underfitting: when the validation/test error deteriorates because the classifier is not flexible enough to fit patterns.
- outliers: points with atypical labels (e.g., rich borrower who defaulted anyway). Increase risk of overfitting.

[In machine learning, the goal is to create a classifier that generalizes to new examples we haven't seen yet. Overfitting and underfitting are both counterproductive to that goal. So we're always seeking a compromise: we want decision boundaries that make fine distinctions without being downright superstitious.]

Kaggle.com:

- Runs ML competitions, including our HWs
- We use 2 data sets:
  public set labels available during competition
  private test set labels known only to Kaggle
  [If your public results are a lot better than your private results, we will know that you overfitted.]